Requested Patent: DE19957235A1

Title:

ARRANGEMENT FOR ACCESSING JAVA APPLICATIONS, CONVERTS ROUTINE CALLS AND DATA INTO JAVA DATA TYPES, CALLS AND RUNS JAVA ROUTINES, GENERATES RESPONSE IN TRANSFER FORMAT, AND TRANSFERS TO APPLICATION ;

Abstracted Patent: DE19957235 ;

Publication Date: 2000-07-20 ;

Inventor(s):

SULZMANN ROBERT (DE); WELSCH MARIN (DE); HERRENDOERFER DIRK (DE) ;

Applicant(s): IBM (US) ;

Application Number: DE19991057235 19991127 ;

Priority Number(s): EP19990100472 19990112 ;

IPC Classification: G06F15/163 ; G06F9/54 ;

Equivalents:

ABSTRACT:

The arrangement contains applications that do not support Java that generate a standard TCP/IP communications call for a routine in a Java class in the database. An arrangement receives the call and associated parameter data, and an arrangement converts the call and data into Java data types. An arrangement calls and runs Java routines using the converted data. An arrangement generates a response to the client application by converting the result data back into a transfer format and an arrangement transfers the data to the non-Java application. An Independent claim is also included for a process for accessing Java routines and a program product stored on a data medium containing the steps of an access process.

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 199 57 235 A 1**

(51) Int. Cl.[7]:
G 06 F 15/163
G 06 F 9/54

(21) Aktenzeichen: 199 57 235.6
(22) Anmeldetag: 27. 11. 1999
(43) Offenlegungstag: 20. 7. 2000

DE 199 57 235 A 1

(30) Unionspriorität:
99 10 0472. 2 12. 01. 1999 EP

(71) Anmelder:
IBM Corp., Armonk, N.Y., US

(74) Vertreter:
Teufel, F., Dipl.-Phys., Pat.-Anw., 70569 Stuttgart

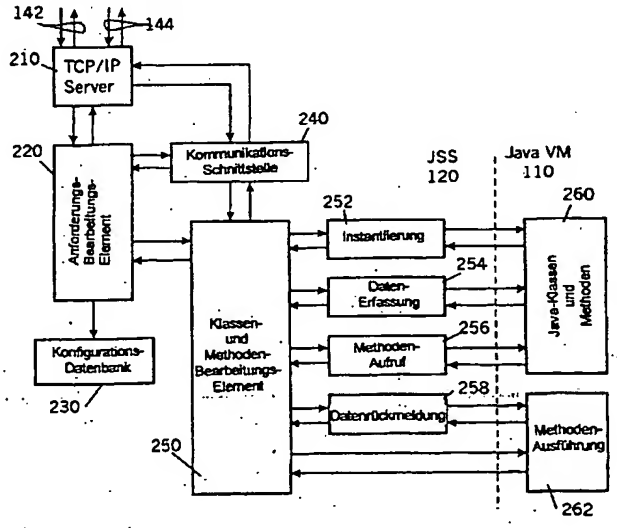
(72) Erfinder:
Herrendörfer, Dirk, 71065 Sindelfingen, DE;
Sulzmann, Robert, 71088 Holzgerlingen, DE;
Welsch, Marin, Dr., 71083 Herrenberg, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Vorrichtung, Prozeß und Produkt für den Zugriff auf Java-Anwendungen

(57) Eine Vorrichtung und ein Prozeß für den Zugriff auf Java-Methoden, die in einer Datenbank mit Java-Klassen und Java-Methoden (260) enthalten sind, durch eine Java nicht unterstützende Anwendung, die auf einem lokalen Rechner (100) oder einem externen Rechner (170) läuft. Die Java nicht unterstützende Anwendung generiert einen Standard-TCP/IP-Kommunikationsaufruf für eine Methode einer Java-Klasse in der genannten Datenbank. Ein Java-Service-Server (120), der auf einer Java VM auf dem lokalen Rechner läuft, empfängt den Methodenaufruf und die zugehörigen Parameterdaten und übernimmt deren Ausführung, einschließlich der Umwandlung des Aufrufs und der zugehörigen Parameterdaten aus einem Transportformat in Java-Ursprungsdatentypen. Die umgewandelten Daten werden verwendet zum Aufrufen einer Java-Methode, die durch Anwendung der genannten Methode auf die umgewandelten Parameterdaten ausgeführt wird. Die Ergebnisdaten der Methodenausführung werden aus dem Java-Format in das Transportformat umgewandelt, in welchem sie an die Java nicht unterstützende Anwendung übermittelt werden.



DE 199 57 235 A 1

Beschreibung

Gebiet der Erfindung

Die Erfindung betrifft eine Vorrichtung für den Zugriff auf Java-Methoden, die in einer Datenbank mit Java-Klassen und Java-Methoden enthalten sind, durch eine Anwendung, wobei die genannte Datenbank in einem lokalen Rechner gespeichert ist und auf sie durch einen Server zugegriffen wird, der in einer Java-Umgebung auf dem genannten lokalen Rechner läuft. Die Erfindung betrifft auch einen Prozeß für den Zugriff auf Java-Methoden und ein auf einem Datenträger gespeichertes Programm-Produkt zur Steuerung eines Rechners, der diesen Prozeß ausführt.

Hintergrund der Erfindung, Stand der Technik

Der Zugriff auf Java-Methoden durch Java nicht unterstützende Anwendungsprogramme erfordert eine Programmierung mit einer detaillierten Schnittstellenspezifikation, die zum Zeitpunkt der Implementierung bekannt sein muß. Außerdem sind Implementierungen dieses Typs speziell auf die Schnittstelle abgestimmt, für die sie programmiert sind. Jede Veränderung der Schnittstelle hat somit eine Neuprogrammierung zur Folge. Außerdem unterstützen einige Programmierungsumgebungen den Java-Zugriff nicht und es müßten daher spezielle Kommunikationsmodule entwickelt werden, um diese Grenze zu überwinden.

Die Verwendung verteilter Java-Anwendungen, bei denen auf einem Rechner laufende Programme Methoden von Java-Objekten aufrufen können, die auf einem anderen Rechner gespeichert sind, ist bekannt. Eine solche entfernte Anwendung wird ermöglichst durch die in dem Buch "JAVA RMI: Remote Method Invocation", T. B. Downing, IDG Books, 1998, ISBN 0-7645-8043-4 beschriebene Schnittstelle Remote Method Invocation (RMI).

Zusammenfassung der Erfindung

Es ist eine Aufgabe der Erfindung, Zugriff auf Java-Methoden durch Java nicht unterstützende Anwendungen auf effektivere Weise und mit weniger Programmieraufwand zu ermöglichen. Außerdem ist es eine Aufgabe der Erfindung, Mittel bereitzustellen, die die Anwendung einer einfachen Schnittstelle für ein Fern-Aufrufen von Java-Methoden durch Java nicht unterstützende Anwendungen über ein Daten-Kommunikationsnetz ermöglichen.

Entsprechend der Erfindung, wie sie in den Ansprüchen definiert ist, generiert eine Java nicht unterstützende Anwendung einen Standard-TCP/IP-Kommunikationsaufruf für eine Methode einer Java-Klasse in einer Datenbank mit Java-Klassen und Java-Methoden. Ein Java-Service-Server empfängt Anforderungen für Methodenaufrufe und verarbeitet diese Aufrufe, einschließlich der Umwandlung des Methodenaufrufs und der zugehörigen Parameterdaten von einem Transportformat in Java-Ursprungs-Datentypen. Die umgewandelten Daten dienen zum Aufrufen einer Java-Methode, die dann durch Anwenden der Methode auf die umgewandelten Parameterdaten ausgeführt wird. Die Ergebnisdaten der Methodenausführung werden aus dem Java-Format in das Transportformat umgewandelt, in dem sie an die Java nicht unterstützende Anwendung übermittelt werden.

TCP/IP hat sich als Standardmittel für die Kommunikation in allen IT-Umgebungen durchgesetzt. Mit der Erfindung hat man Zugriff auf Java-Anwendungen über die TCP/IP-Kommunikation. Der Anwendungsprogrammierer muß nur für die Standard-TCP/IP-Kommunikationsaufrufe programmieren. Für die Kommunikation auf Java-Seite wird ein Java-Service-Server gestartet. Dieser Server erzeugt eine Instanz der Klasse, auf die zugegriffen werden soll, und bildet eine Schnittstelle zu den Ursprungsdatentypen, die zum Aufrufen von Java-Methoden innerhalb einer instantiierten Klasse verwendet werden können.

Die grundlegende Kommunikation zwischen dem Java nicht unterstützenden Client und dem Java-Server-Modul ist textbasiert und somit unabhängig vom Rechner und vom Betriebssystem. Der Server unterstützt außerdem die Übertragung von Hilfeinformationen.

Durch die Verwendung von TCP/IP-Kommunikationsaufrufen für den Aufruf von Java-Methoden wird die Schnittstelle zwischen der Java nicht unterstützenden Anwendung und Java-Ursprungsdatentypen äußerst einfach gehalten. Hierdurch ist es möglich, eine Verknüpfung zu schaffen zwischen Anwendungen, die in low-end Programmiersprachen geschrieben sind, beispielsweise Shell-Skript, Pearl, C oder Visual Basic, und den Java-Anwendungen. Der Server kann von einer externen, Java nicht unterstützenden Anwendung über ein Netzwerk, beispielsweise das Internet, genutzt werden.

Die von dem Server bereitgestellte Schnittstelle ist voll konfigurierbar und wird von einer Konfigurationsdatei angesteuert. Die Konfigurationsdatei kann mit der von den Java-Doc-Dateien bereitgestellten Information des Codes, auf den zugegriffen wird, erstellt werden.

Kurze Beschreibung der Zeichnungen

Im folgenden werden Implementierungen der Erfindung unter Bezugnahme auf Zeichnungen beschrieben. Es zeigt:

Fig. 1 ein Blockschema eines Systems, in dem ein Proxy-Server verwendet wird, der in ein Netz für den Zugriff auf Smart Cards integriert ist, entsprechend dem hierin beschriebenen Ausführungsbeispiel der Erfindung;

Fig. 2 ein Blockschema des Proxy-Servers, der im System der **Fig.** 1 verwendet wird; und

Fig. 3 ein Blockschema von Schritten entsprechend einer Prozeßimplementierung der Erfindung.

Ausführliche Beschreibung eines bevorzugten Ausführungsbeispiels der Erfindung

Fig. 1 zeigt eine virtuelle Java-Maschine (Java VM) **110**, die den Java-Interpreter und das Java-Laufzeitsystem umfaßt und sich auf einem lokalen Rechner **100** befindet, der neben anderen Anwendungsprogrammen installiert ist. Die virtuelle Java-Maschine **110** umfaßt einen Java-Service-Server **120**, bei dem es sich um ein Anwendungsprogramm handelt, das auf Java VM **110** läuft. Die virtuelle Java-Maschine **110** umfaßt weiter eine Datenbank **130**, die eine Vielzahl von registrierten Java-Klassen und Java-Methoden umfaßt. Der Begriff "Methode" hat hier die Bedeutung, wie sie generell in der objektorientierten Technologie verwendet wird, d. h., zur Darstellung einer Funktionsgruppe oder einer Operation. Die in der Datenbank **130** enthaltenen Java-Klassen sind Datensammlungen und Sammlungen von Methoden, die mit diesen Daten arbeiten. Jede Klasse ist einem Anwendungsbereich zugewiesen. Die in der Datenbank **130** enthaltenen Methoden definieren Operationen, die mit den Daten einer Klasse ausgeführt werden können. Eine Klasse kann beispielsweise "Kreis" genannt werden und ihre Methoden führen die Funktionen aus, auf dem Bildschirm einer Anzeigevorrichtung einen Kreis zu zeichnen. Jede Methode ist einer Klasse zugewiesen, wobei jede Klasse über eine Vielzahl von Methoden verfügen kann. Die Methoden können Klassenmethoden sein, die auch als statische Methoden bezeichnet werden, die ohne Instantiierung durch Referen-

zierung dieser Klasse und innerhalb dieser Klasse nach ihren Namen aufgerufen werden können, oder sie können Objektmethoden sein, die aufgerufen werden, nachdem ein Objekt von der referenzierten Klasse instantiiert wurde. Java VM **110** umfaßt außerdem ein Methodenausführungselement **132**, das die von der Datenbank **130** aufgerufenen Methoden ausführt.

Der Java-Service-Server **120** unterstützt einen lokalen Client **140**, bei dem es sich um ein auf dem lokalen Rechner **100** laufendes, Java nicht unterstützendes Anwendungsprogramm handelt. Zu diesem Zweck wird der Client **140** mittels einer Betriebskoppelung **124** mit einem Kommunikationselement **122** des Java-Service-Servers **120** verknüpft. Das Kommunikationselement **122** wird außerdem über einen Kommunikationskanal **144** und ein Netzwerk **150**, beispielsweise das Internet, mit einem externen Client **160** verknüpft, der ein Java nicht unterstützendes Anwendungsprogramm sein kann, das auf einer Workstation **170** läuft, die mit dem Netzwerk über einen Kommunikationskanal **172** verbunden ist. Die Java nicht unterstützenden Anwendungen können Anwendungsprogramme sein, die zum Beispiel in den Programmiersprachen C oder Visual Basic geschrieben sind. Diese Programme nutzen die Java-Methoden in der Datenbank **130**, indem sie während der Laufzeit Aufrufe an diese Methoden ausgeben und diese als Anforderungen an den Java-Service-Server **120** schicken. Die Anwendungsprogramme senden an den Server **120** außerdem Daten, die mit den Aufrufen verbunden sind und auf die die aufgerufenen Methoden angewendet werden sollen. Die Java VM **120** ruft die angeforderten Java-Methoden von der Datenbank **130** ab und führt sie aus, indem sie die genannten verbundenen Daten verwendet, und übergibt nach Berechnung das Ergebnis über den Server **120** zurück an die Client-Anwendung.

Fig. 2 zeigt die Komponenten eines Blockschemas für eine Implementierung des Java-Service-Servers **120**. Die Implementierung umfaßt einen TCP/IP-Server **210**, der der Kommunikationskomponente **122** aus **Fig.** 1 entspricht und den Kommunikationsprotokollstandard TCP (Transmission Control Protocol) und IP (Internet Protocol) erfüllt. Der TCP-IP-Server **210** empfängt Anforderungen über die Kommunikationskanäle **142** und **144** und gibt an den anfordernden Client **140** oder **160** eine Begrüßungsmeldung aus. Der TCP-IP-Server **210** überprüft Benutzername und Passwort der Anforderung und startet, nachdem er eine Verbindung zu dem anfordernden Client hergestellt hat, ein Anforderungsbearbeitungselement **220**, um die von dem Client **140** oder **160** empfangene Anforderung zu verarbeiten. Jede eingegangene Anforderung enthält einen Aufruf für eine Methode in der Datenbank **130**.

Das Anforderungsbearbeitungselement **220** führt einige vorbereitende Operationen durch, die für die Verarbeitung einer Anforderung erforderlich sind. Hierzu gehört die Einleitung einer Konfiguration des Java-Service-Servers **120** mit Hilfe einer Konfigurationsdatenbank **230**, in der alle Informationen, die das Programm für seine Funktion benötigt, gespeichert sind. Außerdem enthält die Datenbank Informationen, die für den Instantiierungsprozeß notwendig sind, sowie Referenzen zu allen Systemmeldungen. Zusätzlich enthält sie Hilfemeldungen, die mit Hilfe des Anforderungsbearbeitungselements **220** und des TCP/IP-Servers **210** dem Benutzer der Anwendung **140** oder **170** auf Wunsch zur Verfügung gestellt werden. Die Konfiguration des JSS-Servers **120** hängt unter anderem von bestimmten Client-Parametern ab, beispielsweise von der Programmiersprache und den von der Client-Anwendung verwendeten Datentypen. Nachdem die Konfiguration abgeschlossen ist, leitet das Anforderungsbearbeitungselement **220** eine Umwandlung des Aufrufs aus einem Transportformat in Java-Ursprungsdaten ein. Die Umwandlungen werden mittels eines Kommunikationsschnittstellenelements **240** durchgeführt. Außer beim Aufrufen einer Klassenmethode oder einer statischen Methode startet das Anforderungsbearbeitungselement **220** über ein Klassen- und Methodenbearbeitungselement **250** eine Instantiierungskomponente **252**, die auf die in der Anforderung angegebene Java-Klasse zugreift und eine Instantiierung der angegebenen Klasse generiert. Diese Klasse ist als eine von vielen Java-Klassen in einer Datenbank **260** enthalten, die der Datenbank **130** der **Fig.** 1 entspricht. Wenn der Aufruf sich auf eine Klassenmethode oder eine statische Methode bezieht, erfolgt der Zugriff auf die entsprechende Methode in der Datenbank **260** ohne eine Instantiierung der von dem Aufruf referenzierten Klasse.

Das Anforderungsbearbeitungselement **220** fordert den anfordernden Client **140** oder **170** zur Angabe der mit dem Aufruf verbundenen Parameterdaten auf. Die Parameterdaten spezifizieren eine auszuführende Zielmethode und enthalten Daten, auf die die Zielmethode angewendet werden soll. Der TCP/IP-Server **210** empfängt die Meldung vom Client und das Anforderungsbearbeitungselement **220** leitet eine Umwandlung der Daten aus dem TCP/IP-Transportformat in Java-Ursprungsdatentypen ein. Ein Datenerfassungselement **254** baut anhand der Spezifikation der Zielmethode und der zu verarbeitenden Client-Daten aus der instantiierten Java-Klasse ein Java-Objekt auf und ein Methodenaufrufelement **256** ruft die Zielmethode in dem genannten Objekt auf. Das aufgerufene Objekt wird dem Methodenbearbeitungselement **250** übergeben, welches das Methodenausführungselement **262** des Java VM **110** startet, um die Zielmethode mit den konvertierten Client-Daten auszuführen.

Bei Abschluß der Methodenoperation leitet ein Datenrückmeldeelement **258** das Ergebnis der Methodenausführung an das Anforderungsbearbeitungselement **220** weiter, das die Ergebnisdaten in das Transportformat umwandelt und sie an den Client **140** oder **160** sendet, von dem der Aufruf ausging. Wenn der Aufruf von dem lokalen Client kam, werden die konvertierten Ergebnisse mit Hilfe des TCP/IP-Servers **210** der Client-Anwendung **140** übergeben. Wenn der Aufruf von dem entfernten Client kam, werden die konvertierten Ergebnisse über den TCP/IP-Server **210** und das Netz **150** an die Client-Anwendung **160** übermittelt.

Zusammenfassend gilt, daß immer dann, wenn ein Methodenaufruf an den Server **120** geht, dieser durch Angabe des Rückmelde-Datentyps, der Anzahl der Parameter und der Datentypen reagiert, die für die Eingangsparameter zu verwenden sind. Der Client teilt die Länge eines jeden Eingabefeldes mit und sendet die entsprechenden Parameterdaten im Textformat. Der Server konvertiert die Textdaten in die gewünschten Java-Datentypen und ruft die gewünschte Java-Methode auf. Nachdem die Ausführung der Methode durch Java VM abgeschlossen ist, wandelt der Server das Ergebnis in das TCP/IP-Textformat um und schickt es, wie oben beschrieben, zurück an den Client.

Die von dem Server **120** bereitgestellte Schnittstelle ist voll konfigurierbar und ist in einer Konfigurationsdatei in der Konfigurationsdatenbank **230** gespeichert. Diese Datei kann mit Informationen aus dem JavaDoc-Tool des Codes erzeugt werden, auf den zugegriffen werden soll. Bei komplexeren Schnittstellen, beispielsweise der Unterstützung von Java-Datentypobjekten, die keine Ursprungsdatentypobjekte sind, ist es möglich, in Java Wrapper-Klassen zu verwenden.

Die Schritte des Anwendungszugriffs- und Anwendungsausführungsprozesses in Java werden unter Bezugnahme auf **Fig.** 3 beschrieben. Der Prozeß betrifft die Verwendung von Objektmethoden, auf die nach Instantiierung einer spe-

zifizierten Java-Klasse zugegriffen werden kann. In Schritt **310** wird eine Anforderung von einem Client empfangen und der Client wird mit dem Java-Service-Server **120** verbunden, der eine Begrüßungsmeldung an den Client schickt. In Schritt **320** gibt der Client eine Java-Klasse an, auf die zugegriffen werden soll, und der Server **120** führt eine Instantiierung dieser Klasse durch. In Schritt **330** gibt der Client die aufzurufende Methode an und der Server **120** fragt nach den Parameterdaten, die von der angegebenen Methode zu verarbeiten sind. In Schritt **340** sendet der Client dem Server **120** Parameterdaten; dieser baut mit den umgewandelten Parameterdaten ein Java-Objekt auf. In Schritt **350** ist das Senden der Parameterdaten beendet und der Server **120** ruft die Zielmethode auf. In Schritt **360** führt Java VM die Zielmethode aus. Der Server **120** empfängt die Methoden-Rückmeldedaten, die ausgewertet, in das Transportformat umgewandelt und an die Client-Anwendung gesendet werden. In Schritt **370** empfängt der Client die Antwort und verarbeitet diese bei der Ausführung seines Anwendungsprogramms. Mit Schritt **380** beendet der Client seine Verbindung zum Server **120** oder startet eine neue Anforderung. Im folgenden werden Beispiele einer Kommunikation zwischen einem Client und einem Server beschrieben.

Erstes Beispiel

Der Client leitet die Verbindung zum Server ein. Der Server kann mit einer Begrüßungsmeldung antworten:

>"200 JavaServiceServer (C) IBM 1998 READY."

Der Client fordert die Methode an, auf die er zugreifen möchte:

<"initialize"

Die Methode "initialize" ist eine Methode mit der Definition (void)initialize(void). Der Server sendet die Antwort auf den Aufruf zurück:

>"200 Methode erfolgreich aufgerufen – keine Rückmeldung."

Dies ist eine einfache Form zum Aufrufen einer Methode. Komplizierter ist der Aufruf, wenn der Transport zusätzlicher Daten von der und zu der Anwendung erforderlich ist.

Zweites Beispiel

In diesem Beispiel benötigt die aufgerufene Methode zwei Strings und meldet eine ganze Zahl zurück.

Wieder wird die Methode aufgerufen:

<"verify"

Der Server antwortet mit den benötigten Datentypen:

>"400 (int)verify(String,String)"

Der Client muß dem Server mitteilen, wie viele Daten er senden möchte:

>"10 10"

Der Server bestätigt:

>"200 OK. Beginne mit dem Senden der Daten."

Der Server sendet ein Datenpaket mit beiden Informationsfeldern:

<"HELLOWWORLDHELLOWWORLD"

Der Client meldet zurück, daß die gesendeten Daten korrekt waren und antwortet mit dem Senden von Rückmeldedaten:

>"200 Methode erfolgreich aufgerufen – sende Rückmeldung."
"2"
"20"

Der Server **120** bestätigt die Anforderung und weist darauf hin, daß er zwei Bytes Daten zurücksenden wird; anschließend sendet er die zwei Datenbytes. Die Definition des Datenfelds wird auf die eingegangenen Daten angewendet und die Anwendung **140** oder **160** kann jetzt ihre eigene Darstellung für den empfangenen Wert verwenden.

Zwar wurde die Erfindung unter Bezugnahme auf eine bevorzugte Implementierung der Erfindung beschrieben, jedoch liegen Änderungen oder andere Ausführungsbeispiele der Erfindung im Erfindungsumfang, wie er in den Ansprüchen definiert wird.

Patentansprüche

1. Eine Vorrichtung zum Zugriff auf Java-Methoden, die in einem Datenbank-Mittel mit Java-Klassen und Java-Methoden (**130, 260**) enthalten sind, durch eine Anwendung, wobei das genannte Datenbank-Mittel in einem lokalen Rechner (**100**) gespeichert ist und Server-Mittel (**120**), die in einer Java-Umgebung auf dem genannten lokalen Rechner laufen, darauf zugreifen, umfassend:
   (a) Java nicht unterstützende Anwendungsmittel (**140**), die einen Standard-TCP/IP-Kommunikationsaufruf für eine Methode einer Java-Klasse in der genannten Datenbank (**130, 260**) generieren;
   (b) Mittel (**122, 210**) für den Empfang des genannten Methodenaufrufs und der zugehörigen Parameterdaten von dem genannten Anwendungsmittel;
   (c) Mittel (**220, 240**) zum Umwandeln des genannten Methodenaufrufs und der genannten zugehörigen Parameterdaten von dem genannten Anwendungsmittel aus einem Transportformat in Java-Ursprungsdatentypen;
   (d) Mittel (**250, 256**) zum Aufrufen der genannten Java-Methode und zum Ausführen der Methode unter Verwendung der genannten umgewandelten Parameterdaten;
   (e) Mittel (**258, 220, 240**) zum Generieren einer Antwort an die Client-Anwendung (**140**) durch Umwandeln der Ergebnisdaten der Ausführung vom Java-Format in ein Transportformat; und
   (f) Mittel (**210**) zur Übergabe der umgewandelten Ergebnisdaten an das Java nicht unterstützende Anwendungsmittel.

2. Eine Vorrichtung nach Anspruch 1, umfassend TCP/IP-Servermittel (**210**) zum Empfangen der genannten Methodenaufrufe und der genannten Parameterdaten von der genannten Anwendung (**140**) und für die Übertragung der genannten Ergebnisdaten an die genannte Anwendung (**140**).

3. Eine Vorrichtung nach Anspruch 1, weiter umfassend Mittel zur Erzeugung einer Instanz (**252**) der in dem genannten Aufruf angegebenen Java-Klasse, die

die aufzurufende Methode enthält.

4. Eine Vorrichtung nach Anspruch 3, weiter umfassend Mittel (**254**) zur Generierung eines Objekts aus einer instantiierten Klasse, das eine Methode enthält, auf die durch das genannte aufrufende Mittel (**250, 256**) zugegriffen wird.

5. Eine Vorrichtung nach Anspruch 1, umfassend einen Java-Service-Server (**130**), der eine auf einer virtuellen Java-Maschine (**110**) auf dem genannten lokalen Rechner (**100**) laufende Anwendung ist, für die Durchführung des Zugriffs auf eine Methode in der genannten Java-Datenbank (**130, 260**) und zur Ausführung der genannten Methode mit Daten, die von der Java nicht unterstützenden Anwendung empfangen wurden, wobei der genannte Java-Service-Server (**130**) das genannte TCP/IP Servermittel (**210**) umfaßt.

6. Eine Vorrichtung nach Anspruch 5, bei der der Java Service-Server (**130**) eine Schnittstelle (**240**) umfaßt, um Methodenaufrufe und zugehörige Datenparameter aus einem Transportformat in ein Java-Format umzusetzen und um Ergebnisdaten aus der Ausführung der Methode aus einem Java-Format in ein Transportformat umzusetzen.

7. Eine Vorrichtung nach Anspruch 5, bei der der Java Service-Server (**130**) die Übertragung der Hilfemeldungen an die genannte Anwendung durchführt.

8. Eine Vorrichtung nach einem der Ansprüche 1–7, bei der die genannte Java nicht unterstützende Anwendung (**160**) auf einem externen Rechner (**170**) installiert ist, der über ein Datenkommunikationsnetzwerk (**150**) mit dem genannten empfangenden und übergebenden Mittel (**210**) verbunden ist.

9. Ein Prozeß für den Zugriff auf Java-Methoden, die in einer Datenbank mit Java-Klassen und Java-Methoden (**130, 260**) enthalten sind, durch eine Anwendung (**140**), wobei die genannte Datenbank in einem lokalen Rechner (**100**) gespeichert ist und ein Server, der in einer Java-Umgebung (**120**) auf dem genannten lokalen Rechner läuft, auf sie zugreift, gekennzeichnet durch folgende Schritte:

(a) Erzeugen eines Standard-TCP/IP-Kommunikationsaufrufs für eine Methode einer Java-Klasse in der genannten Datenbank (**320**) durch eine Java nicht unterstützende Anwendung (**140**);

(b) Empfangen des genannten Methodenaufrufs und der zugehörigen Parameterdaten von der genannten Anwendung (**330, 340**);

(c) Umwandeln des genannten Methodenaufrufs und der zugehörigen Parameterdaten aus einem Transportformat in Java-Ursprungsdatentypen (**340**);

(d) Aufrufen der genannten Java-Methode zur Ausführung durch Verwenden der genannten umgewandelten Parameterdaten (**350, 360**);

(e) Generieren einer Antwort an die Client-Anwendung (**140**), durch Umwandeln der Ergebnisdaten der Methodenausführung in ein Transportformat (**360**); und

(f) Übergabe des umgewandelten Ergebnisses an die Java nicht unterstützende Anwendung.

10. Ein Prozeß nach Anspruch 9, umfassend den Schritt (d1) des Erzeugens einer Instanz der in dem genannten Aufruf spezifizierten Java-Klasse, welche die aufzurufende Methode enthält.

11. Ein Prozeß nach Anspruch 10, umfassend den Schritt (d2) des Erzeugens eines Objekts aus einer instantiierten Klasse, welches die Methode enthält, auf die von dem genannten aufrufenden Mittel (**250, 256**) zugegriffen werden soll.

12. Ein Prozeß nach einem der Ansprüche 9–11, bei dem Schritt (a) während der Laufzeit der Anwendung ausgeführt wird.

13. Ein Prozeß nach Anspruch 9, umfassend den Schritt des Übertragens von Hilfemeldungen an die genannte Anwendung.

14. Ein Prozeß nach einem der Ansprüche 9–13, bei dem der genannte Methodenaufruf und die genannten, damit verbundenen Parameter an den genannten Empfangsschritt (b) von einer auf einem entfernten Rechner (**170**) laufenden Anwendung (**160**) über ein Datenkommunikationsnetz (**150**) übertragen werden.

15. Ein Programmprodukt, das auf einem Datenträger gespeichert ist, umfassend die Schritte eines Prozesses für den Zugriff auf Java-Methoden, die in einer Datenbank mit Java-Klassen und Java-Methoden (**130, 260**) enthalten sind, durch eine Anwendung (**140**), wobei die genannte Datenbank in einem lokalen Rechner (**100**) gespeichert ist und auf sie von einem Server zugegriffen wird, der in einer Java-Umgebung auf dem genannten lokalen Rechner läuft, gekennzeichnet durch folgende Schritte:

(a) Generieren eines Standard-TCP/IP-KOmmunikationsaufrufs für eine Methode einer Java-Klasse in der genannten Datenbank (**320**) durch eine Java nicht unterstützende Anwendung;

(b) Empfangen des genannten Methodenaufrufs und der zugehörigen Parameterdaten von der genannten Anwendung (**330, 340**)

(c) Umwandeln des genannten Methodenaufrufs und der zugehörigen Parameterdaten von einem Transportformat in Java-Ursprungsdatentypen (**340**);

(d) Aufrufen der genannten Java-Methode zur Ausführung, unter Anwendung der genannten konvertierten Parameterdaten (**350, 360**);

(e) Generieren einer Antwort an die Client-Anwendung (**140**), durch Umwandeln der Ergebnisdaten der Methodenausführung in ein Transportformat (**360**); und

(f) Übergabe des umgewandelten Ergebnisses an die Java nicht unterstützende Anwendung.

16. Ein Programmprodukt nach Anspruch 15, umfassend den Schritt (d1) des Erzeugens einer Instanz der Java-Klasse, die in dem genannten Aufruf spezifiziert wurde und die die aufzurufende Methode enthält.

17. Ein Programmprodukt nach Anspruch 16, umfassend den Schritt (d2) des Generierens eines Objekts aus einer instantiierten Klasse, die die Methode enthält, auf die von dem genannten aufrufenden Mittel (**250, 256**) zugegriffen werden soll.

18. Ein Prozeß nach einem der Ansprüche 15–17, bei dem Schritt (a) während der Laufzeit der Anwendung ausgeführt wird.

19. Ein Programmprodukt nach Anspruch 15, umfassend den Schritt des Übertragens von Hilfemeldungen an die genannte Anwendung.

20. Ein Programmprodukt nach einem der Ansprüche 15–19, bei dem der genannte Methodenaufruf und die zugehörigen Parameter über ein Datenkommunikationsnetz (**150**) von einer Anwendung (**160**), die auf einem entfernten Rechner (**170**) installiert ist, an den genannten Empfangsschritt (b) übertragen werden.

Hierzu 3 Seite(n) Zeichnungen

Nummer: DE 199 57 235 A1
Int. Cl.[7]: G 06 F 15/163
Offenlegungstag: 20. Juli 2000

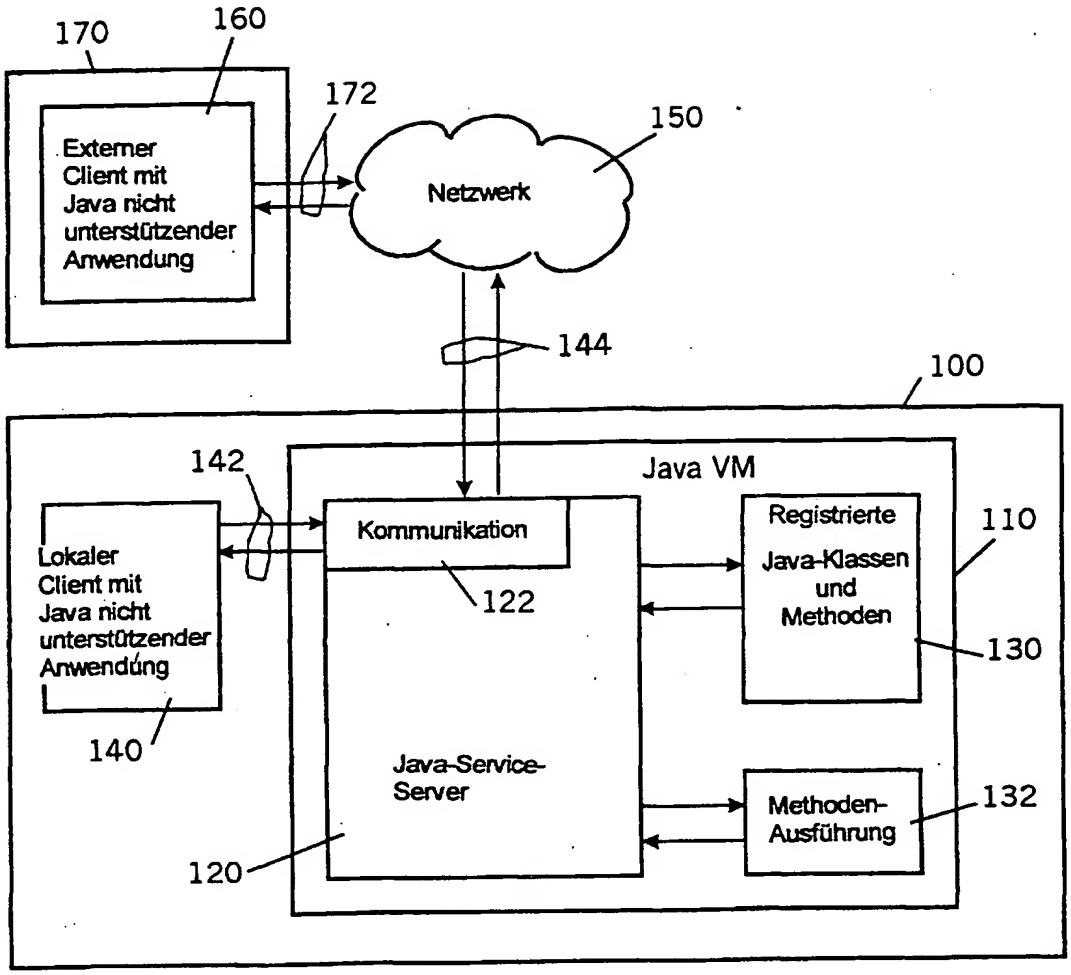


FIG. 1

Nummer: DE 199 57 235 A1
Int. Cl.[7]: G 06 F 15/163
Offenlegungstag: 20. Juli 2000

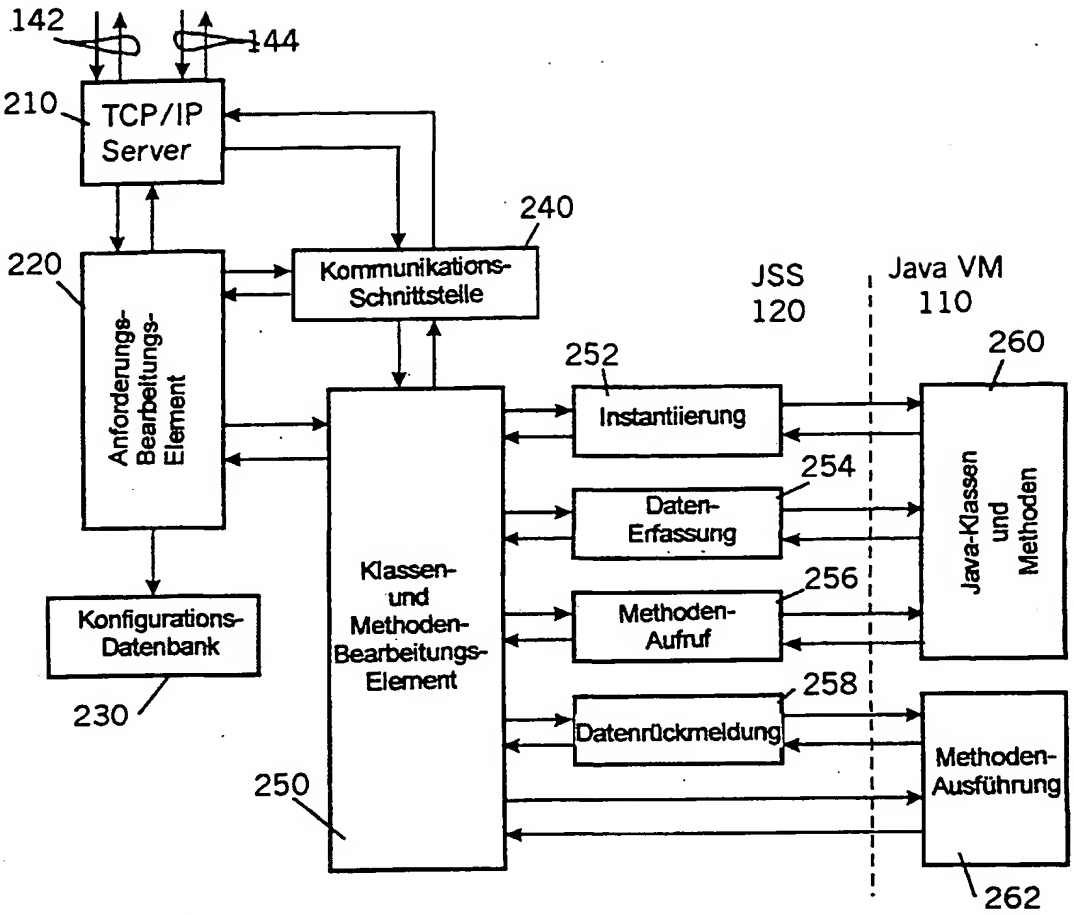


FIG. 2

Nummer: DE 199 57 235 A1
Int. Cl.[7]: G 06 F 15/163
Offenlegungstag: 20. Juli 2000

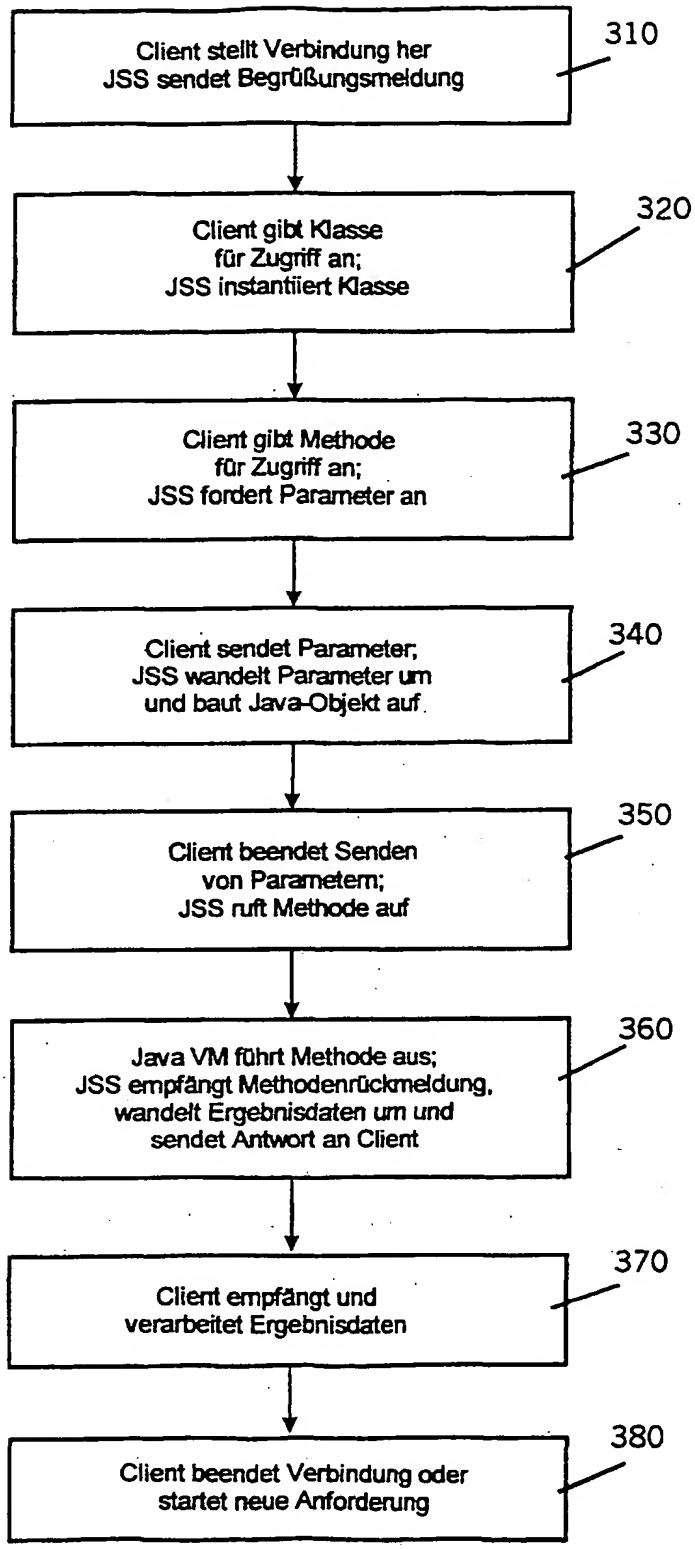


FIG. 3